HAYES, VICTORIA, M.A. The Evolution of Cooperation: A Recreation of Axelrod's Computer Tournament. (2017)
Directed by Dr. Jan Rychtář. 70 pp.

The iterated Prisoner's Dilemma is a commonly studied game in Game Theory. Many real life situations, such as trench warfare during World War I, can be modeled by such a game. Robert Axelrod implemented a computer tournament in order to determine the best strategy during repeated interactions. Various entries, ranging from very simple to very sophisticated strategies, competed in his tournament. We recreate the tournament using the programming language Matlab and examine the results. Although our results are not entirely identical to Axelrod's results, we confirm Axelrod's general findings. In particular, in order for a strategy to be successful, it should be nice, forgiving, relatively easy to understand by its opponents and also retaliatory.

THE EVOLUTION OF COOPERATION: A RECREATION OF AXELROD'S COMPUTER TOURNAMENT

by

Victoria Hayes

A Thesis Submitted to
the Faculty of The Graduate School at
The University of North Carolina at Greensboro
in Partial Fulfillment
of the Requirements for the Degree
Master of Arts

Greensboro
2017

Approved by

____
Committee Chair

To my Lord for giving me the strength to do what I could not do by myself.

APPROVAL PAGE

This thesis written by Victoria Hayes has been approved by the following committee of the Faculty of The Graduate School at The University of North Carolina at Greensboro.

Committee Chair ____________________________________
Jan Rychtář

Committee Members ____________________________________
Sat Gupta

____
Sebastian Pauli

____
Date of Acceptance by Committee

____
Date of Final Oral Examination

ACKNOWLEDGMENTS

I would like to express my gratitude to my supervisor Dr. Jan Rychtář for the useful comments, remarks and engagement through the learning process of this master thesis. Furthermore, I would like to thank my committee members, Dr. Sat Gupta and Dr. Sebastian Pauli for taking the time to read my thesis, and for their help and support. Also, I would like to thank my peers for their continual positive reinforcement as I worked on writing the code for our tournament. I would like to thank my loved ones, who have supported me throughout entire process. I will be grateful forever for your love.

TABLE OF CONTENTS

Page

LIST OF TABLES vi

CHAPTER

I. AN INTRODUCTION TO GAME THEORY 1

1.1. Basic Definitions 1
1.2. Prisoner's Dilemma 3

II. FIVE RULES FOR THE EVOLUTION OF COOPERATION 6

2.1. Kin Selection 7
2.2. Direct Reciprocity 7
2.3. Indirect Reciprocity 9
2.4. Network Reciprocity 9
2.5. Group Selection 10

III. AXELROD'S ORIGINAL TOURNAMENT 12

3.1. Background to Axelrod's Tournament 12
3.2. The Computer Tournament 13
3.3. The Collective Stability of TFT 18
3.4. Live-and-Let-Live in WWI 20

IV. RESULTS OF OUR COMPUTER TOURNAMENT 25

V. CONCLUSIONS 33

REFERENCES 34

APPENDIX A. A RECREATION OF AXELROD'S TOURNAMENT 35

APPENDIX B. DESCRIPTIONS OF STRATEGIES IN AXELROD'S TOURNAMENT 67

LIST OF TABLES

Page

Table 1. Payoff Matrix for Rock-Paper-Scissors . . . 3

Table 2. Payoff Matrix for Tucker's Prisoner's Scenario . . . 4

Table 3. Payoff Matrix for Axelrod's Tournament . . . 14

Table 4. Tournament Results: First Tournament . . . 15

Table 5. Best Results . . . 27

Table 6. Worst Results . . . 29

Table 7. Average Results . . . 31

CHAPTER I

AN INTRODUCTION TO GAME THEORY

1.1 Basic Definitions

Conflict has been widespread throughout the whole of human history. When two or more individuals have different values or goals, they will compete for control over the events, and thus conflict appears. *Game theory* uses mathematics to study such situations. Its study was greatly motivated in 1944 by the publication of *Theory of Games and Economic Behavior* by John Von Neumann and Oskar Morgenstern [TCB09]. We begin with some basic definitions:

Definition 1.1 ([Sta99])**.** A *game* is said to be a situation or conflict between individuals.

Definition 1.2. The participants in a game are called *players*.

While there are numerous types of games that model interactions between individuals, we limit our discussion to 2-player games. In particular, we will focus primarily on a 2-player game known as the *Prisoner's Dilemma*. This game will be discussed in detail in a section 1.2.

Just as when you sit down to play a board game with your friends, players in a game must have a strategy to follow in order to win the game. In game theory, a player is not said to win or lose the game, but rather a strategy can be successful or unsuccessful toward a particular goal.

Definition 1.3. In game theory, a *strategy* is a specification of what to do in any given situation.

The success of a player's strategy in a given game is measured by a *payoff*. The payoff is equivalent to the score that a player earns in a particular game. The payoff is often represented in a *payoff matrix*.

A very common two player game is Rock-Paper-Scissors. Consider two players Ruth and Charlie. Saul Stahl [Sta99] gives an explicit description of this childhood game:

> Ruth and Charlie face each other and simultaneously display their hands in one of the following three shapes: a fist denoting a *rock*, the forefinger and middle finger extended and spread to as to suggest *scissors*, or a downward facing palm denoting a sheet of *paper*. The rock wins over the scissors since it can shatter them, the scissors win of the paper since they can cut it, and the paper wins over the rock since it can be wrapped around the latter.

The payoff matrix for Rock-Paper-Scissors game is shown in Table 1. For each time the game is played, each player will earn either one point for winning, lose one point for losing, or earn zero points in the case of a tie. The payoffs are represented by ordered pairs. The first coordinate of the ordered pair is the payoff for the row player (in this case, Ruth) and the second coordinate is the payoff of the column player (Charlie). For example, in the first row and second column of the payoff matrix we see the ordered pair (-1, 1), which is the payoff for when Ruth plays *rock* and Charlie plays *paper*. The -1 in the ordered pair indicates that Ruth earns a score of -1 because she loses when she plays *rock* against *paper*. Analogously, the 1 in the ordered pair tells that Charlie earns 1 point because *paper* beats *rock*.

Table 1. Payoff Matrix for Rock-Paper-Scissors: Here the ordered pair (-1, 1) in the *rock* row and the *paper* column indicates a payoff of -1 to Ruth and 1 to Charlie provided that Ruth Played *rock* and Charlie played *paper*.

		Charlie		
		Rock	**Paper**	**Scissors**
Ruth	**Rock**	(0,0)	(-1, 1)	(1, -1)
	Paper	(1, -1)	(0, 0)	(-1, 1)
	Scissors	(-1, 1)	(1, -1)	(0, 0)

Notice that the sum of each ordered pair in Table 1 is zero. This indicates that Rock-Paper-Scissors is a *zero-sum game*. In a zero-sum game, one players win is the other player's loss. Not all games are zero-sum games. In *nonzero-sum games*, the payoff may be a measurable amount as in zero-sum games or it may be something abstract such as *one-upmanship*, which is a loss of face [Sta99].

1.2 Prisoner's Dilemma

We discuss a particular nonzero-sum game, the *Prisoner's Dilemma*.

Definition 1.4. A nonzero-sum game is said to be *non-cooperative* if the players do not communicate with each other about ways and methods to improve their payoff [Sta99].

The Prisoner's Dilemma game is a non-cooperative, nonzero-sum game. This game was first given its name by a Princeton mathematician, Albert W Tucker, in 1950 [TCB09]. Consider the following scenario:

Table 2. Payoff Matrix for Tucker's Prisoner's Scenario: Here the ordered pair (-5, -5) in the *Keep Quiet* row and the *Keep Quiet* column indicates a payoff of -5 to both prisoners, where the negative represents the length of the prison sentence.

	Keep Quiet	**Testify**
Keep Quiet	(-5, -5)	(-15, 0)
Testify	(0, -15)	(-10, -10)

> You and a partner are arrested and held in connection with a certain robbery. There is not enough evidence to convict you of armed robbery, but the authorities separate you and your partner for questioning in hopes that you will confess to the armed robbery. You have the choice to testify against your partner for a reduced sentence or remain quiet.

Table 2 is the payoff matrix for the scenario. The payoff is in terms of the length of the prison sentence, where the sentence is represented by a negative integer. As described in the scenario, you have two choices: keep quiet (cooperate) or testify (defect). No matter what your partner chooses, your payoff will be greater if you choose to defect. However, overall, the payoff is better if both cooperate. Hence, the dilemma!

We can study the interaction of persons who are not prisoners, and this can still be modeled by a Prisoner's Dilemma game similar to the scenario above. The value of the payoff will be different depending on the specifics of the game. Regardless of the actual values of the payoff, certain factors remain the same:

- There is a *reward* for mutual cooperation.
- There is a *sucker's payoff* for the player who cooperates when the opponent defects.

- There is a *temptation to defect* for the player who defects when the opponent cooperates.

If you will only meet your opponent once in such a game, then it pays to take advantage of the cooperation of your opponent and defect. However, you run the risk of your opponent implementing the same strategy, and then both players will be punished with a lesser payoff.

When two players engage with each other more than once in a row, and the players are able to remember the previous moves of the other player, the game becomes an *iterated Prisoner's Dilemma.* While the overarching principles are the same, a more complex strategy may be needed.

CHAPTER II

FIVE RULES FOR THE EVOLUTION OF COOPERATION

Martin A. Nowak proposes five rules for the evolution of cooperation: kin selection, direct reciprocity, indirect reciprocity, network reciprocity, and group selection [Now06]. We discuss these in more detail below. In the subsequent chapters, we then put more emphasis on the direct reciprocity.

As will be seen in later sections, Nowak gives a simple rule for each strategy that designates whether natural selection can lead to cooperation. Each rule is based on certain parameters. The two most important parameters are those of *cost* and *benefit*. One who cooperates pays a certain cost so that another individual may receive a benefit. A person who is a defector will have no cost and will not pay out any benefits. Cost and benefit are measured for each individual involved in terms of fitness. In a mixed population of defectors and cooperators, it is evident that defectors will have a higher average fitness than cooperators because they pay out no benefits to others. In a mixed society, the cooperators may fade from the picture eventually, leading to a population of defectors. In pure, unmixed populations, the population of cooperators has the highest average fitness, and the population of defectors has the lowest. Thus, while it may benefit an individual to defect in a mixed society, this defection will most likely lead to the eventual disappearance of cooperators. Then, that society will no longer be mixed, and will be a society of defectors, with the lowest level of fitness. This is not conducive for the evolution of the population. Therefore, cooperation is the preferable strategy for the continuation of society.

2.1 Kin Selection

This first rule stems from the idea that natural selection will favor cooperation if the individuals involved are genetic relatives. This rule of interaction is known as Hamilton's Rule. Hamilton's rule takes into account a new parameter called *relatedness*. Relatedness is the probability of sharing a gene. For example, the probability that two brothers share a gene is $\frac{1}{2}$ and the probability of two cousins sharing a gene is $\frac{1}{8}$. So, we see that this rule is motivated by "selfish genes" that wish to propel themselves [Daw16]. In order for individuals to cooperate with this strategy, the relatedness must be greater than the cost-to-benefit ratio of the one paying the benefit. Thus, natural selection will tend toward cooperation with this rule:

$$r > \frac{c}{b} \tag{2.1}$$

where r is the relatedness, c is the cost of the cooperation and b is the benefit of cooperation.

2.2 Direct Reciprocity

Kin selection applies only to interaction of relatives. While it is a viable rule for the evolution of cooperation in such a population of relatives, it is not sufficient to only consider such relationships. Direct reciprocity is a mechanism for the evolution of cooperation among individuals who are not related. This mechanism works best in a scenario of repeated encounters between two individuals, where each individual has the choice to cooperate or defect, otherwise known as the Prisoner's Dilemma as discussed in Section 3.1. In Axelrod's computer tournaments simulating such games of interaction, he found that a strategy of direct reciprocity called TFT was the best strategy [Axe84]. TFT always begins with cooperation, and then it does whatever the other player did in the previous round. Simple though it is, this strategy came out on top for both of Axelrod's tournaments. No strategy is perfect, and so TFT has its own weaknesses. TIT FOR TAT cannot correct any mistakes. For example, if the other player accidentally defects, this may lead to a long line of retaliation from the player using the TFT strategy. A slight variation of TFT, the GENEROUS TIT FOR TAT strategy allows the player to cooperate sometimes following a defection. This idea of forgiveness is crucial to move toward cooperation. In time, TFT was replaced by an even simpler rule of engagement, WIN-STAY, LOSE-SHIFT. This rule says that you will repeat your previous move when you are "winning," but you will change your move otherwise. With these two rules of direct reciprocity, TFT is still effective at leading toward cooperation in a society where most individuals are defectors. However, once cooperation is established, the best rule to follow is WIN-STAY, LOSE-SHIFT.

Regardless of the actual strategy being used, direct reciprocity may lead natural selection to the evolution of cooperation if the probability of another encounter with the same two individuals is high enough. This probability is denoted by w. Again, this probability must exceed the cost-to-benefit ratio. Natural selection will favor cooperation with the rule:

$$w > \frac{c}{b} \tag{2.2}$$

2.3 Indirect Reciprocity

Direct reciprocity is a good rule to follow when there are repeated interactions among the same individuals. However, it is more likely for interactions among people to be fleeting. In direct reciprocity, both individuals must be able to provide help. With indirect reciprocity, one person is in a position to help another individual, but the individual receiving the benefit has not opportunity to reciprocate the act. This can be seen in society in our donations to charities. The fuel behind indirect reciprocity is *reputation*. When one person helps another, it establishes a good reputation for the donor. This good reputation is noted by others in the population, and it may be rewarded by others. As a result, individuals will tend toward cooperation if the probability of knowing someone's probability is good enough. The probability of knowing one's reputation is denoted by q. This rule seems like a selfishly motivated rule of operation, and in fact it is. Indirect reciprocity will only promote the evolution of cooperation if the following rule is satisfied:

$$q > \frac{c}{b} \tag{2.3}$$

2.4 Network Reciprocity

The argument has been made that natural selection will tend toward defection in a mixed population [Now06]. This conclusion is based on the idea that everyone in the population interacts equally with every other member in the society. While this is possible, it is not likely to happen in human populations. Most populations are not well-mixed. This leads to another evolutionary approach to analyzing these interactions–evolutionary graph theory. In this approach, the individuals in a society are represented by the vertices of the graph. The edges represent the interactions with others. In the simplest of terms with cooperators and defectors, we see that cooperators pay a cost for the neighbor to receive a benefit. Defectors pay no cost and their neighbors receive no benefits. In these terms, cooperators will form network clusters, and so cooperation prevails. This is network reciprocity. Network reciprocity introduces another parameter into the equation, and that is the average number of neighbors that an individual has. The average number of neighbors is called k. For natural selection to lead to cooperation, the benefit-to-cost ratio must be greater than the average number of neighbors. Hence, we see cooperation with this simple rule:

$$\frac{b}{c} > k \tag{2.4}$$

2.5 Group Selection

Thus far, we have viewed natural selection as it acts upon individuals. In turn, the individuals shape society. Selection also acts on groups as a whole. This method for the evolution of cooperation uses a simple model of society divided in different groups. Cooperators will help others in their group. Defectors help no one. An individual reproduces proportional to their payoff. Offspring are added to the same group. Groups may split in two if the population of the group reaches a certain size. As a result of the creation of this new group, another group will become extinct to limit the total population size. As a result, there is competition between groups because certain groups grow faster than others, and thus split more often. As a general rule, pure cooperator groups grow faster than pure defector groups. In mixed groups, individuals who defect will increase faster than cooperators. This may eventually lead to the group becoming pure defectors. Using this simple model, letting n be the maximum group size and m is the number of groups, we find another simple rule for the evolution of cooperation:

$$\frac{b}{c} > 1 + \frac{n}{m} \tag{2.5}$$

CHAPTER III

AXELROD'S ORIGINAL TOURNAMENT

3.1 Background to Axelrod's Tournament

Interactions among individuals–whether the individuals are cells, animals, or humans–occur all the time. These relationships have been studied across many disciplines. In Prisoner's Dilemma game, these individuals have two choices: cooperate or defect. The innate tendencies of individuals are to be selfish. This selfishness may lead to cooperation or defection depending on the payoff to the individual. Studies have indicated that cooperation leads to the better payoff for all involved over time [Axe84, Now06]. From an evolutionary perspective, cooperation is imperative if the natural evolution process will construct new levels of organization [Now06]. While all societies are based on cooperation, human society is the one society that engages in the most complex games of interaction. In the lens of natural selection, competition is the leader in motivation for behaviors, and this competition naturally opposes cooperation. Nevertheless, it holds true that cooperation is necessary to construct the new levels of organization in society, and so there must be some strategies that will push individuals to cooperation.

In 1980, Robert Axelrod implemented a project that stemmed from one simple question: When should a person cooperate or be selfish in an ongoing interaction with another person? This type of situation can be represented by an iterated Prisoner's Dilemma game.

Many real life situations may be modeled by such a game, and Axelrod set out to determine the best strategy to use in such situations. He invited experts in game theory to submit programs for a computer Prisoner's Dilemma tournament. Fourteen entries were sent in as contenders in Axelrod's computer tournament. In the first tournament, he ran the fourteen entries and a random rule against each other and determined a winner. After the initial tournament, the results were circulated and another tournament took place. The same strategy surfaced as the winner again. The winner for both tournaments was a program called TIT FOR TAT (TFT), which is a strategy that begins with cooperation, and thereafter returns what the other player did on the previous move. The specifics of the tournament are discussed in the following section.

3.2 The Computer Tournament

Axelrod's computer Prisoner's Dilemma tournament set out to determine how to choose effectively in an iterated Prisoner's Dilemma situation. His tournament was set up as a round robin, where each entry was paired with each other entry. Each entry was also paired with its twin and with RANDOM, which was a strategy that chose randomly to cooperate or defect with equal probability. Each game involved 200 moves. The payoffs were as follows:

- Mutual cooperation resulted in both players earning the *reward* of 3 points.
- Mutual defection resulted in both players earning the *punishment* of 1 point each.

Table 3. Payoff Matrix for Axelrod's Tournament: Here the ordered pair (3, 3) in the *Cooperate* row and the *Cooperate* column indicates a payoff of 3 to both players.

	Cooperate	**Defect**
Cooperate	(3, 3)	(0, 5)
Defect	(5, 0)	(1, 1)

- If one player cooperated and the other defected, the one who cooperated would earn 0 points – known as the *sucker's payoff*, while the one who defected would earn 5 points – known as the *temptation* to defect.

TFT was the simplest of all the programs submitted to the tournament, and it proved to be the best overall. In a second tournament, other entries were submitted that were based upon TFT, but even with their attempts to improve it, TFT still won. However, all of the strategies that were top runners in the tournament had something in common with TFT. The best strategies share the property of being *nice*. A strategy is nice if it is never the first to defect, or to say it will not be the first to defect before the last few moves. See Table 4 for the results from the first tournament.

Table 4. Tournament Results: First Tournament: Here the number 214 in the Joss row and the Tideman indicates Joss's score when playing a game with 200 moves against Tideman. Other numbers are to be interpreted similarly.

Strategy	**TFT**	**Tideman**	**Nydegger**	**Grofman**	**Shubik**	**Stein & Rap**	**Friedman**	**Davis**	**Graaskamp**	**Downing**	**Feld**	**Joss**	**Tullock**	**Unnamed**	**Random**	*Average Score*
TFT	600	595	600	600	600	595	600	600	597	597	280	225	279	359	441	504
Tideman	600	596	600	601	600	596	600	600	30	601	271	213	291	455	573	500
Nydegger	600	595	600	600	600	595	600	600	433	158	354	374	347	368	464	486
Grofman	600	595	600	600	600	594	600	600	376	309	280	236	305	426	507	482
Shubik	600	595	600	600	600	595	600	600	348	271	274	272	265	448	543	481
Stein & Rap	600	596	600	602	600	596	600	600	319	200	252	249	280	480	592	478
Friedman	600	595	600	600	600	595	600	600	307	207	235	213	263	489	598	473
Davis	600	595	600	600	600	595	600	600	307	194	238	247	253	450	598	472
Graaskamp	597	305	462	375	348	314	302	302	588	625	268	238	274	466	548	401
Downing	597	591	398	289	261	215	202	239	555	202	436	540	243	487	604	391
Feld	285	272	426	286	297	255	235	239	274	704	246	236	272	420	467	328
Joss	230	214	409	237	286	254	213	252	244	634	236	224	273	390	469	304
Tullock	284	287	415	293	318	271	243	229	278	193	271	260	273	416	478	301
Unnamed	362	231	397	273	230	149	133	173	187	133	317	366	345	413	526	282
Random	442	142	407	313	219	141	108	137	189	102	360	416	419	300	450	276

Each of the top eight rules in the tournament were nice rules. The factor that sets apart the top eight entries was their interaction with strategies that were not nice. There were two strategies called *kingmakers* that made the biggest difference among the top eight entries in the tournament.

DOWNING is the most important kingmaker. It focuses on "outcome maximization" [Axe84]. The reasoning behind DOWNING is very different from that of TFT. It is based on understanding what the other player will decide to do. If the other player seems responsive to the choices that DOWNING makes, then DOWNING will cooperate. On the other hand, if the other player does not seem to be responsive to DOWNING's choices, then it will lean toward the advantage that comes from defecting. To make these decisions about the responsiveness of the other player, DOWNING estimates two different conditional probabilities: the probability that the other will cooperate given that DOWNING cooperates, and the probability that the other will cooperate given that DOWNING defects. It then chooses the probability that will maximize the long term payoff. Since DOWNING has no conditional probabilities to begin with, it begins with an initial assumption that the other player will be unresponsive. This forces it to defect for the first two moves. Depending on the other strategy, DOWNING could be doomed to punish itself, or against certain opponents, such as TFT, it learns to be cooperative. In our recreation of the tournament, this error is corrected, and we implement a revised version of DOWNING that does not defect on the first two moves, but rather it begins with the assumption that the opponent will be responsive.

Another important factor in the success of a nice rule is the idea of *forgiveness.* Forgiveness is the idea of cooperating following a defection by the other player. The nice rules that were least forgiving did not do as well as TFT. One such entry that was lacking in forgiveness was FRIEDMAN. FRIEDMAN is a totally unforgiving rule that uses permanent retaliation. It will never be the first to defect, but once the other player defects, it will defect every time. In comparison to the winner, TFT is unforgiving for one move, but then it is totally forgiving of that defection.

While TFT reigned supreme in the tournament, there do exist certain strategies that were not in the tournament that could have won had they entered [Axe84].

TIT FOR TWO TATS is a strategy that defects only if the other player had defected on the two previous turns. This makes it more forgiving than TFT, and it proves that being more forgiving contributes to a higher payoff.

LOOK AHEAD was used in Axelrod's preliminary tournament, and was the winner of that preliminary. LOOK AHEAD is a rule that is inspired by techniques used in artificial intelligence programs in playing chess.

There is one unlikely contender for the top spot in the tournament, and that is a slight variation on DOWNING. If it had begun with an initial assumption that the other player would be responsive instead of unresponsive, then DOWNING could have been a winner of the tournament. However, as it is, DOWNING is a pessimistic rule, and it therefore suffers the consequences.

In the second round of the tournament, there were strategies that used a controlled number of defections. These "not nice" strategies were big indicators in the level of success of the nice strategies. Two such strategies were TESTER and TRANQUILIZER. TESTER is written to exploit the other player. It always defects for the first move. If the other player ever defects, TESTER apologizes by cooperating for the next move, and then plays TFT for the remaining moves of the game. Otherwise, it cooperates on the second and third moves and then defects for every move afterward. As a result, TESTER does not score well, but it does do a good job at exploiting some of the nicer rules.

TRANQUILIZER is a rule that is somewhat sneaky. Initially, it tries to establish a mutually rewarding relationship. Once the rewarding relationship has been established, it will try to exploit the other player. TRANQUILIZER will cooperate for the first couple dozen moves as long as the other player is cooperating. Then, it will attempt an unprovoked defection. TRANQUILIZER will never defect twice in a row, and it will not defect more than one-fourth of the time.

While it was shown that TFT was the winner of Axelrod's two computer tournaments, that does not guarantee that it is the best strategy to employ in every situation. Through strong testing including hypothetical tournaments, TFT proved itself to be the winner again. Also, through ecological tests, TIT FOR TAT remained at the top of the list. Consequently, it can be said that TFT is a *robust* strategy. That is, it would be successful in a wide variety of environments. The reasons for TFT's robust nature stem from its combination of niceness, forgiveness, retaliation, and clarity. Clarity allows the other player to recognize it for what it is, and appreciate its behavior and lack of exploitability.

3.3 The Collective Stability of TFT

Axelrod's computer tournament indicated that TFT would thrive as a strategy. It follows that eventually all players might adopt the strategy. If this happened, would there ever be a need to use an alternative strategy? If an alternative rule is able to infiltrate a population using a single strategy, and it is able to score higher than the population average, then that alternative rule is said to have invaded the population. If a particular strategy cannot be invaded, it is said to be *collectively stable*. Thus arises the question of the stability of TFT. Axelrod states a proposition about the collective stability of TFT.

Proposition 3.1 ([Axe84]). *TIT FOR TAT is collectively stable if and only if the probability of the game ending is small enough.*

The current move in a game always carries more weight than a future move because there is no guarantee of a future move. Now, we are faced with deciding what is "small enough." Axelrod discovered that if the probability to end is $\frac{1}{3}$ or smaller, then TFT is collectively stable. If the probability grows to over $\frac{1}{2}$, then TFT is no longer stable, and it would be best to defect every move [Axe84].

Through analysis of Axelrod's tournament, there are four suggestions for how to do well in an iterated Prisoner's Dilemma:

(1) Don't be envious (Forgive).

(2) Don't be the first to defect (Be nice).

(3) Reciprocate both cooperation and defection (Retaliate).

(4) Don't be too clever (Have clarity).

It is clear that TFT abides by all four of those guidelines, and so it is easy to see why it is such an effective strategy. Therefore, implementation of a strategy similar to TFT can lead to the evolution of cooperation in a population.

3.4 Live-and-Let-Live in WWI

During World War I, along the Western front in Europe, a level of cooperation emerged among members of opposing armies. Trench warfare was very common, and along the Western front there were many gruesome battles. However, in between battles, a philosophy developed among the soldiers. Soldiers in opposing armies could be clearly seen walking within shooting distance behind their respective lines, yet no one was shot. The men in those trenches had adopted a "live-and-let-live" philosophy. This policy among soldiers thrived, despite all the efforts of Senior officers. The idea of "live-and-let-live" contradicts military logic. The cooperation between enemies persisted when it should have never existed in the first place.

While it may not appear to be as such, the interaction between two small units in a quiet section along the Western front is a Prisoner's Dilemma. Each unit is a player and the choices in the game are to shoot to kill or to shoot in a manner that does not inflict damage. The dilemma stems from the fact that if a major battle should arise, one army would want the enemy's army to be weakened prior to battle. Thus, when looking at the short term goals, it may be wise to shoot to kill whether the enemy is returning fire or not. This leads to the idea that mutual deflection may be ideal in the short term. This mutual deflection is better for an army than unilateral restraint, unless it is the opponent's army that restrains [Axe84]. As a result, both sides would prefer mutual restraint to random acts of aggression between the units.

The different units interacted with each other for extended periods of time. So, just as in the earlier discussion of interaction among players, while defection may benefit a player in the short term, strategies develop for interaction over an extended period of time. What we have here is the evolution of cooperation between the players. The choices and behaviors of the units in the trenches in World War I support the expected outcomes from game theory. Just as TFT was a successful strategy that implemented cooperation that was based upon reciprocity, the "live-and-let-live" strategy operates in the same manner. Both sides would mutually restrain themselves and keep from shooting to kill. If there was a defection, and one army caused the death of some soldiers, then the opposing army would retaliate causing damage that was comparable or sometimes slightly more devastating. Then, the two armies would slip back into a state of cooperation.

Where did such cooperation first develop among enemies? The early battles of World War I were very mobile and very destructive. As time progressed, the enemy lines stabilized. The result was trenches along the front lines for opposing armies with an empty "no man's land" in between the front lines. According to diary entries of soldiers, the cooperation developed quite spontaneously in many different places along the Western front. The initial cases of such cooperation are connected with common meal times. It became obvious to the soldiers in the trenches that the enemies across the way must have been partaking in a similar routine at the same time because things were quiet on both sides. Eventually, communication began between the units, and truces were made. One such famous verbal truce was the Christmas truce during the first Christmas in the trenches. However, such verbal truces were quickly and easily punished. Other factors, such as inclement weather contributed to the evolution of cooperation among enemies. Certain weather conditions made it impossible to shoot at each other. If that condition lasted long enough, then the cooperation sometimes continued after the weather cleared. Ultimately, the biggest contributor to the development of the "live-and-let-live" mentality was the idea of self-preservation. Soldiers knew that their enemies shared the same needs as they did. The armies learned that a unilateral attack would just result in retaliation from the other side. However, restraint on one side would most likely result in restraint on the other side. Then everyone involved would be able to live to fight another day.

Once started, the cooperation among enemies could easily spread from troop to troop, down the line. One reason that cooperation was so sustainable was because opposing armies made it clearly known that they could retaliate if necessary. In a sense, each army would "flex their muscles" in an attempt to prove that they were a worthy opponent that should not be reckoned with. The strategy of "live-and-let-live" continued on in the trenches even as battalions would change out since the soldiers moving out of the trench were familiar with the soldiers moving in the trench. The agreements and policies would be passed along like a legacy to the new soldiers who would occupy the front lines.

The "live-and-let-live" policy could not last forever (else, we would still be in World War I). Military officials instituted a type of attack known as the raid. A raid was a carefully planned attack on an enemy's trench. A successful raid would collect prisoners, while an unsuccessful raid would collect bodies. Either way, there would be evidence of an attack. Unlike when soldiers could pretend to shoot to kill, when in fact they were shooting to avoid inflicting damage, soldiers could not pretend to implement a raid. Thus, this new strategy quickly brought an end to the camaraderie in the trenches.

When examining the "live-and-let-live" strategy, cooperation did not evolve through blind mutation or survival of the fittest. This strategy developed as the result of conscious decisions made by the players to cooperate on the basis of reciprocation [Axe84]. The strategy did not thrive because of survival of the fittest because even with a poor strategy in place, soldiers could easily be replaced, and the unit would still remain in its location in the trenches. The surviving presence of the players on both sides had nothing to do with the particular strategies implemented by those on the front lines.

The "live-and-let-live" strategy follows the theory of the evolution of cooperation, but there are two new developments that arise from this particular method: ethics and ritual. In time, the interactions between the two opposing units led to the development of concern for the fellow human being. Soldiers did not want to violate an agreement of trust, nor did they want to see another person hurt. Through extended interaction, the values and payoffs changed for the players. After persistent cooperation, the payoff for this mutual cooperation became higher than it initially was for the units. The raids brought out more ethics among the players. A soldier feels an obligation to retaliate for a fallen comrade, and so revenge resulted from the raids. Revenge drove soldiers to retaliate.

The other development that follows from "live-and-let-live" is the development of rituals. What is meant by rituals in the trenches? Since both sides agreed not to shoot to kill, the use of artillery was limited and used in a manner that would be less than effective. Additionally, the smaller arms were used more often in warfare. Also, different armies would follow a regular schedule when attacking targets. This allowed the opposing army to know when and where the attack would take place so that the army could protect its soldiers and equipment from such an attack. Another purpose for the rituals was to satisfy the higher military authorities. Such attacks appeared to be aggressive acts of war, and so the superiors were satisfied. However, with such precision and regularity, the attacks were a beacon of peace to the enemy army on the other side.

CHAPTER IV

RESULTS OF OUR COMPUTER TOURNAMENT

In the Appendix A, we include our Matlab code for the recreation of Axelrod's computer tournament. We have written two different programs: one that replicates the computer tournament, and one that analyzes the results from the tournament. The code for the actual computer tournament is modified from a file written by Mark Broom and Jan Rychtář [BR13]. The descriptions for the strategies originate from *The Axelrod Library* [KCH+17]. Dr. Jan Rychtář and Dr. Sebastian Pauli contributed to the writing and revision of these programs.

In Section A.1 we include the code that is a recreation of Axelrod's original computer tournament. The code includes all fourteen of the original entries in the tournament, as well as the RANDOM strategy. Our computer tournament is a round robin tournament, where each game consists of 200 moves. In the program, you are given the option to determine how many times you wish to play the tournament. We played the tournament 1,000 times.

There are some slight variations in our code from the original computer tournament. The most notable change is to the DOWNING strategy. In our code, we write a revised DOWNING strategy. In the original strategy, DOWNING was a pessimistic strategy that assumed that its opponent would not be cooperative. Because of this fact, DOWNING did not perform well in the first computer tournament. In Table 4, we see that DOWNING came in tenth place overall. Axelrod suggested that if DOWNING initially assumed that its opponent was cooperative, then it could potentially be the winner of the computer tournament [Axe84]. We followed this suggestion and made the appropriate adjustments to our DOWNING rule.

Section A.2 includes the code used to perform data analysis on the computer tournaments that were played using the code in Section A.1. The code produces three different spreadsheets: *best*, *worst*, and *average*. The *best* spreadsheet displays the best scores earned against each opponent in each of the 1,000 tournaments. It also includes the best average score that was attained by each strategy in the tournament. In addition, the *best* spreadsheet gives the best overall ranking for each strategy. This is useful because it allows us to easily see how the strategies ranked against each other. Similarly, the *worst* and *average* spreadsheets give the worst scores and average scores, respectively.

The results from our computer tournament differ from Axelrod's tournament in several ways. TFT did not do as well in our tournament. In Table 5 we see that the best that TFT ranked in any of the 1,000 tournaments was second place. TFT even did as poorly as ninth place, and on average TFT ranked at about 6.7 out of 15. TFT's lack of success in our computer tournament may be attributed to a couple of factors. Several of the competing strategies make decisions about cooperation and defection at some given probability. That element of chance can greatly affect the outcome when playing a game versus TFT. In effect, one could argue that TFT was simply lucky in Axelrod's two computer tournaments. To be more accurate, TFT's success in Axelrod's first tournament is due largely in part to the *kingmakers*[RSC15]. DOWNING was a major *kingmaker* in the original tournament, and with the newly revised DOWNING, this characteristic is changed.

Table 5. Best Results: Here the number 594 in the Downing row and the Stein column means that out of all 1,000 runs of the tournament, the best score that Downing achieved while playing against Stein was 594 points. Also, the best average score that Downing earned in all 1,000 tournaments was 548.7 and the best ranking was 1. Other numbers are to be interpreted similarly.

	TFT	Tideman	Nydegger	Grofman	Shubik	Stein	Grim	Davis	Graaskamp	Downing	Feld	Joss	Tullock	Unnamed	Random	Average	Order
TFT	600	600	600	600	600	595	600	600	525	600	244	210	561	469	490	518.1	2
Tideman	600	600	600	600	600	595	600	600	596	600	260	211	565	568	616	535.9	1
Nydegger	600	600	600	600	600	594	600	600	582	600	378	108	579	312	377	503.5	5
Grofman	600	600	600	600	600	595	600	600	650	600	373	172	559	422	505	523.3	2
Shubik	600	600	600	600	600	595	600	600	740	600	260	217	410	560	630	532.6	1
Stein	600	600	604	604	600	596	600	600	522	604	239	210	588	605	655	536.1	1
Grim	600	600	600	600	600	595	600	600	303	600	235	207	373	599	693	507.9	4
Davis	600	600	600	600	600	595	600	600	303	600	231	209	371	580	681	509.3	4
Graaskamp	525	586	656	614	551	522	303	303	610	658	349	146	574	559	629	478.5	10
Downing	600	600	600	600	600	594	600	600	588	600	393	209	582	593	674	548.7	1
Feld	249	296	840	621	302	244	235	236	669	798	227	209	281	535	589	399	11
Joss	215	284	984	680	282	222	210	238	744	257	212	209	267	617	677	389.5	11
Tullock	566	560	668	611	430	578	363	351	632	666	249	206	468	487	529	410.6	11
Unnamed	469	405	880	631	280	429	156	177	705	589	325	195	436	492	564	384.5	11
Random	495	472	848	617	269	485	141	154	679	802	353	183	474	454	554	379.8	13

In Table 6, we see that DOWNING's worst average score in our 1,000 tournaments was 511.8. In particular, the worst that DOWNING ever scored in a game occurred when playing against JOSS, and DOWNING scored only 198 points. Even so, the worst that DOWNING ever did in our tournament was fourth place. In fact, DOWNING's best average score in our computer tournaments was approximately 549. The benchmark for success for a *good* strategy in a game with 200 moves is a score of 600. A score of 600 is achieved when both players cooperate with each other on every move. The success of the revised DOWNING confirms Axelrod's assumption that DOWNING would become a contender for the top spot in the tournament if it was altered to become more optimistic. With a slight revision, DOWNING transforms from a tenth place *kingmaker* to a tournament champion.

Table 6. Worst Results: Here the number 525 in the TFT row and the Graaskamp column means that out of all 1,000 tournaments, the worst score that TFT achieved while playing against Graaskamp was 525 points. Also, the worst average score that TFT earned in all 1,000 tournaments was 490.5 and the worst ranking was 9. Other numbers are to be interepreted similarly.

	TFT	Tideman	Nydegger	Grofman	Shubik	Stein	Grim	Davis	Graaskamp	Downing	Feld	Joss	Tullock	Unnamed	Random	Average	Order
TFT	600	600	600	600	600	595	600	600	525	600	203	201	224	348	402	490.5	9
Tideman	600	600	600	600	600	595	600	600	563	600	205	183	216	350	393	497	8
Nydegger	600	600	600	600	600	594	600	600	516	600	240	24	498	175	228	482.9	9
Grofman	600	600	600	600	600	594	600	600	557	600	239	92	390	254	336	497.5	8
Shubik	600	600	600	600	600	595	600	600	607	600	201	183	220	401	472	509.1	5
Stein	600	600	604	600	600	596	600	600	522	604	205	199	224	359	415	498.7	7
Grim	600	600	600	600	600	595	600	600	303	600	203	201	225	423	491	490.7	9
Davis	600	600	600	600	600	595	600	600	303	600	202	193	225	421	509	489.5	9
Graaskamp	525	555	612	465	324	522	303	303	527	608	224	77	378	244	300	432.3	10
Downing	600	600	600	600	600	594	600	600	513	600	199	198	501	357	297	511.8	4
Feld	208	264	748	523	263	210	207	221	566	214	203	201	225	389	441	347.1	14
Joss	206	271	928	587	271	210	205	228	652	205	204	201	234	413	494	368.2	13
Tullock	229	286	614	500	286	229	219	218	449	612	203	192	227	369	412	362.1	14
Unnamed	348	198	764	507	199	131	104	135	183	108	227	123	275	326	380	306.5	15
Random	402	186	732	465	186	114	82	111	174	89	236	102	304	296	370	289.5	15

There were other strategies that were able to win the tournament at least once in our computer simulations. SHUBIK and STEIN performed very well in our computer tournament. In the original tournament, SHUBIK came in fifth place. Table 7 shows that SHUBIK's average ranking is approximately 2.3. At the worst, SHUBIK matched its performance in Axelrod's tournament and came in fifth. Similarly, STEIN was much improved in our computer tournament. STEIN followed SHUBIK in the original tournament with a ranking of sixth place. While STEIN did earn a seventh place spot in its worst performance of the tournaments, it still averaged about 3.7 out of 15. These results make both SHUBIK and STEIN good candidates for rules in an iterated Prisoner's Dilemma situation. Both performed better than TFT. Interestingly, both SHUBIK and STEIN are variations of TFT. SHUBIK is less forgiving and more retaliatory than TFT. STEIN checks for a random strategy, and takes advantage of the randomness of its opponent by defecting. Otherwise, STEIN remains cooperative.

Table 7. Average Results: Here the number 600 in the Grim row and the Davis column means that out of all 1,000 tournaments, the average score that Grim achieved while playing against Davis was 600 points. Also, the average of all average scores that Grim earned in all 1,000 tournaments was 499.01 and the average ranking was 6.9. Other numbers are to be interpreted similarly.

	TFT	Tideman	Nydegger	Grofman	Shubik	Stein	Grim	Davis	Graaskamp	Downing	Feld	Joss	Tullock	Unnamed	Random	Average	Order
TFT	600	600	600	600	600	595	600	600	525	600	217.267	201.627	305.983	407.344	448.491	500.0475	6.677
Tideman	600	600	600	600	600	595	600	600	582.904	600	233.815	190.774	281.798	459.635	544.971	512.5931	3.702
Nydegger	600	600	600	600	600	594	600	600	553.488	600	309.66	63.285	543.834	241.426	301.277	493.798	8.63
Grofman	600	600	600	600	600	594.752	600	600	597.345	600	299.99	130.661	466.991	346.2	407.719	509.5772	4.406
Shubik	600	600	600	600	600	595	600	600	668.951	600	233.754	190.7	267.094	477.017	545.918	518.5623	2.339
Stein	600	600	604	600.992	600	596	600	600	522	604	216.969	200.181	305.938	464.915	578.529	512.9016	3.663
Grim	600	600	600	600	600	595	600	600	303	600	214.528	201.59	256.508	517.536	597.048	499.014	6.982
Davis	600	600	600	600	600	595	600	600	303	600	216.894	196.182	258.406	507.554	584.78	497.4544	7.555
Graaskamp	525	572.664	631.008	552.655	433.976	522	303	303	572.492	631.04	288.057	111.395	486.103	397.015	549.596	458.6001	10
Downing	600	600	600	600	600	594	600	600	553.44	600	275.855	201.364	543.675	510.86	563.35	536.1696	1.046
Feld	222.267	278.13	793.56	574.435	278.509	222.059	216.078	223.874	616.992	560.29	210.563	201.542	242.265	459.681	518.614	374.5906	12.252
Joss	206.627	276.369	957.81	638.701	276.33	214.191	206.085	236.067	701.065	207.404	206.452	202.408	240.533	507.411	584.683	377.4757	12.236
Tullock	310.978	335.553	637.444	553.756	318.439	310.143	239.398	240.516	573.578	637.55	220.795	195.308	288.729	424.905	469.874	383.7977	11.575
Unnamed	410.259	277.5	835.126	567.935	230.187	261.94	126.091	155.544	437.115	144.345	276.596	151.901	364.315	407.291	479.608	341.7169	14.017
Random	451.011	228.601	795.157	548.954	226.833	150.579	106.988	135.135	220.476	184.845	288.034	138.048	403.059	380.643	449.197	313.8373	14.92

The results from our computer tournament parallel those in Axelrod's tournament in numerous ways. On average, the *nice* strategies still ranked in the top eight in our tournaments. A few of the nice strategies did slip into ninth place for at least one of the tournaments, but this makes sense because DOWNING made a big move from tenth place into the top four places. There is also a clear distinction between the scores of the *nice* strategies in comparison to the scores of the *not nice* strategies. The ranking of the remaining *not nice* rules performed similarly in our tournaments as they did in Axelrod's tournament. The RANDOM and UNNAMED strategies still remained in the bottom two spots on average. However, both of these strategies were able to improve slightly at least once in the 1,000 tournaments, with UNNAMED placing in eleventh place once and RANDOM moving into thirteenth place. GRAASKAMP's performance in our computer tournament is comparable to the original tournament. Actually, GRAASKAMP's performance was the most consistent of all the strategies. In each of the 1,000 tournaments played, GRAASKAMP always came in tenth place. We can argue that GRAASKAMP performed exactly the same in our computer tournament as it did in the original tournament. While GRAASKAMP did slide back one position into tenth place, we have to remember that DOWNING was much improved, and so DOWNING's improved performance should shift all the other players back.

CHAPTER V

CONCLUSIONS

We have seen in Chapter II that there are several potential mechanisms for the evolution of cooperation: kin selection, direct reciprocity, indirect reciprocity, and group selection. In this thesis, we have focused on direct reciprocity as implemented in the repeated interactions between individuals. In an effort to better understand the best rules to follow when in an iterated Prisoner's Dilemma, we have slightly modified Axelrod's original computer tournament to simulate the evolution of cooperation using Matlab. We have found that several strategies, most notably DOWNING, STEIN and SHUBIK, did perform actually much better than what Axelrod's results suggested and, surprisingly, TFT performed worse than in the tournament. At the same time, however, STEIN and SHUBIK are simply variations of TFT. Thus, we can conclude, that in order to be successful in the iterated Prisoner's Dilemma, the strategy should have the following characteristics:

- Forgive.
- Be nice.
- Retaliate.
- Be clear.

REFERENCES

[Axe84] Robert M Axelrod, *The evolution of cooperation*, Basic books, 1984.

[BR13] Mark Broom and Jan Rychtář, *Game-theoretical models in biology*, CRC Press, 2013.

[Daw16] Richard Dawkins, *The selfish gene*, Oxford university press, 2016.

[KCH+17] Vince Knight, Owen Campbell, Marc Harper, James Campbell, Karol M. Langner, VSN Reddy Janga, Nikoleta, Thomas Campbell, Jason Young, Geraint Palmer, Kristian Glass, Malgorzata Turzanska, Martin Jones, Cameron Davidson-Pilon, Sourav Singh, Ranjini Das, Aaron Kratz, Timothy Standen, Paul Slavin, Adam Pohl, andy boot, WonkySpecs, Jochen Müller, Georgios Koutsovoulos, Tomáš Ehrlich, Karl, Luis Visintini, Martin Chorley, Brice Fernandes, and ABarriscale, *Axelrod-python/axelrod: v2.6.0*, `https://doi.org/10.5281/zenodo.322624`, February 2017.

[Now06] Martin A Nowak, *Five rules for the evolution of cooperation*, Science **314** (2006), no. 5805, 1560–1563.

[RSC15] Amnon Rapoport, Darryl A Seale, and Andrew M Colman, *Is Tit-For-Tat the answer? on the conclusions drawn from Axelrod's tournaments*, PloS one **10** (2015), no. 7, e0134128.

[Sta99] Saul Stahl, *A gentle introduction to game theory*, American Mathematical Society, 1999.

[TCB09] Alan D. Taylor, Bruce P. Conrad, and Steven J. Brams, *For all practical purposes: mathematical literacy in today's world*, W.H. Freeman and Company, 2009.

APPENDIX A

A RECREATION OF AXELROD'S TOURNAMENT

A.1 Matlab Code for Tournament

```
function IPD_Tournament
% implementation of a round robin tournament of iterated PD game
% User specifies the PD payoff matrix
% User also specifies the list of strategies and their definition

%% User defined parameters
PD_payoff =[1,5;
    0,3]; %payoff matrix for PD game


%%Init and auxiliar variables
strategy = {@TFT, @Tideman, @Nydegger, @Grofman, @Shubik, @Stein, ...
   @Grim, @Davis, @Graaskamp, @Downing, @Feld, @Joss, @Tullock, ...
   @Unnamed, @Random};% list of strategies
strategy_names = {'TFT', 'Tideman', 'Nydegger', 'Grofman', ...
   'Shubik', 'Stein', 'Grim', 'Davis', 'Graaskamp', 'Downing', ...
   'Feld', 'Joss', 'Tullock', 'Unnamed' 'Random'};


Defect = 1;
Cooperate = 2;
%with the above notation, PD_payoff(Defect, Cooperate)
%determines the payoff to a player that defected if the other
```

```
%cooperated
Nplayers = length(strategy); % how many players entered
score = zeros(1,Nplayers);    % init of scores as 0

%% init counters for the strategies

% Shubik counters
Retaliation_Counter = 0; % init the count of retaliation
Moves_to_retaliate = 0; % not retaliating anymore

%Downing
C_count = 0;    %count of my own cooperations
D_count = 0;    %count of my own defections
DC_count = 0;  %count of opponent cooperations after my defection
CC_count = 0;   %count of opponent cooperations after my cooperation
good = 1;   %probability opponent is responsive
bad = 0;    %probability opponent is unresponsive

%Stein
Stein_move_counter = 0; %initialize the move counter
Opponent_is_random = 0; %assume opponent is not random

%Feld
probability_to_cooperate = 1;

%Tideman
opp_D_counter = 0; %initializ the opponent deflection counter
last_refresh_round = -20;

```

```

%% User defined functions specifying strategies
% functions take my and opponent's history of the moves
% as an input and produce my move as an output

    function move=TFT(My_hist, Opp_hist)
        %Tit for Tat Strategy
        %Always cooperates on the first move.  After the first
        %move, it reciprocates the opponent's last move.

        if isempty(My_hist) % if first move
            move=Cooperate; % cooperate on the first move
        else % not first move
            move=Opp_hist(end); % repeat opponent's last move
        end;
    end

    function move=Grim(My_hist, Opp_hist)
        %Grim Stategy
        %This strategy was known as Grudger by Friedman in the
        %Axelrod Tournament.
        %This strategy will cooperate until the opponent defects.
        %Then, it will always defect for all of the remaining moves.

        if any(Opp_hist==Defect) % if opponent ever defected
            move=Defect;
        else
            move=Cooperate;
        end;
```

```
    end

    function move=Random(My_hist, Opp_hist)
        %Random Strategy
        %Cooperates and defects on a completely random basis, not
        %dependent on the opponent's moves.

        move = randi(2); %randomly choose between cooperate and
                         %defect
    end

    function move=Grofman(My_hist, Opp_hist)
        %Grofman Strategy
        %It cooperates on the first two moves, and then returns the
        %opponent's moves for the next five moves (i.e. It
        %cooperates on the first move and then plays TFT for moves
        %2-6). For the remaining moves of the game, it cooperates if
        %both it and the opponent made the same move in the previous
        %round.  Otherwise,it cooperates randomly with a probability
        %of 2/7.

        if isempty(My_hist) %if first move
            move=Cooperate; %cooperate on first move
        else %not first move
            if (length(My_hist))<6 % moves 2 through 6
                move=TFT(My_hist, Opp_hist);%play TFT for moves
                                            %2 through 6
            else %not moves 2-6
                if My_hist(end)==Opp_hist(end) %if the previous
```

```
                                           %move is the same
                                           %for both players
                move=Cooperate; %Cooperate on the next move
            else %if the previous move is not the same for
                 %both players
                if rand()≤ 2/7
                    move=Cooperate; %cooperate randomly with
                                    %prob 2/7
                else %the other 5/7 of the time
                    move=Defect;%defect
                end
            end
        end;
    end;
end

function move=Davis(My_hist, Opp_hist)
    %Davis Strategy
    %Cooperates on the first 10 moves, then it plays
    %Grim (Friedman) for the remaining moves of the game.

    if length(My_hist)<10 %for the first 10 moves
        move=Cooperate; %Cooperate on first 10 moves
    else %after the first 10 moves
        move=Grim(My_hist, Opp_hist); %Play Grim after first
                                      %10 moves
    end;
end
```

```
function output = ISRANDOM(Opp_hist)
    %takes a sequence of 1's and 2's and returns 1 if the
    %sequence can be random

    %This function will be used with the Graaskamp strategy
    %below.

    L = length(Opp_hist);
    C_count_random = sum(Opp_hist)-L;  %defect is 2, coop is 1
    if  ((C_count_random < (L/2 - ...
            3*sqrt(L/4)))||(C_count_random > (L/2 + 3*sqrt(L/4))))
        %takes the number of 1's and checks against binomial
        %distribution; if outside of usual bounds, it is not
        %random
        output = 0;
    else
        %can be random
        %check pairs

        %initialize
        CC_count_random = 0;
        CD_count_random = 0;
        DC_count_random = 0;
        DD_count_random = 0;
        for i = 1:L-1
            pair = Opp_hist(i:i+1);
            if pair == [1 1]
                CC_count_random = CC_count_random +1;
            elseif pair == [1 2]
```

```
                CD_count_random = CD_count_random +1;
            elseif pair == [2 2]
                DD_count_random = DD_count_random +1;
            elseif  pair == [2 1]
                DC_count_random = DC_count_random +1;
            end
        end
        %all counts should be roughly 1/4 of the L-1 pairs
        %               CC_count
        %               CD_count
        %               DC_count
        %               DD_count
        %               L
        %               L/4 - 3*sqrt(L*3/16)
        %               L/4 + 3*sqrt(L*3/16)
        L=L-1; %here we have only L-1 pairs
        if  ((CC_count_random < (L/4 - ...
            3*sqrt(L*3/16)))||(CC_count_random > (L/4 + ...
            3*sqrt(L*3/16)))) ...
                || ((CD_count_random < (L/4 - ...
                    3*sqrt(L*3/16)))||(CD_count_random > (L/4 ...
                    + 3*sqrt(L*3/16)))) ...
                || ((DC_count_random < (L/4 - ...
                    3*sqrt(L*3/16)))||(DC_count_random > (L/4 ...
                    + 3*sqrt(L*3/16)))) ...
                || ((DD_count_random < (L/4 - ...
                    3*sqrt(L*3/16)))||(DD_count_random > (L/4 ...
                    + 3*sqrt(L*3/16))))
            %distribution; if outside of usual bounds, it is
```

```
            %not random
            output = 0;
        else
            output = 1;
        end
    end
end

function output = DO_I_KNOW_THIS_STRATEGY(My_hist, Opp_hist)

    %compares the sequence of moves of the opponent to any
    %known strategy (other than random), returns 1 if I know
    %this strategy

    %take all strategies we know
    %strategy = {@TFT, @Grim, @Random, @Tester, @Grofman,
    %@Davis, @Graaskamp, @Joss, @Tideman, @Nydegger, @Shubik,
    %@Stein, @Downing, @Feld, @Tullock, @Unnamed};

    %This function will be used within the Graaskamp strategy
    %below.

    known_deterministic_strategies = [1,7,8] ;

    output = 0; %start with the hypothesis that I do not know
    %the strategy
    for str = known_deterministic_strategies
        m=1; %start the hypothetical moves
        %go all the way to the end of the history or to the
```

```
            %place where I deviate from TFT
            while (m<length(Opp_hist)) && (Opp_hist(m) == ...
                strategy{str}(Opp_hist(1:m-1), My_hist(1:m-1)))
                m = m+1;
            end
            if m== length(Opp_hist) %if I got all the way to the
                                        %end it means I played like
                                        %TFT
                output = 1; %it can be strategy that I know
            end %there is no if, if I know it
                %can be one, it could be the other, but I
                %will never assign 0 to output once I test
                %for a strategy
        end
    end

    function move=Graaskamp(My_hist, Opp_hist)
        % Graaskamp Strategy
        % Plays TFT for 50 rounds, defects on round 51, plays TFT
        %for rounds 52-56, a check is then made to see if the
        %opponent is playing randomly, if so it defects for the
        %rest of the rounds. The strategy also checks to see if
        %the opponent is playing some other strategy that it
        %recognizes. If so, it plays TFT for the remaining moves
        %of the game.Otherwise, if the opponent is not is
        %playing a recognizable strategy, it cooperates and
        %randomly defects every 5 to 15 moves. The last bit will
        %be addressed by randomly defecting with probability 0.1

```

```
M = length(My_hist)+1; %denotes the current round
if  M≤ 50 %for the first 50 rounds
    move=TFT(My_hist, Opp_hist);%Play TFT for first 50 rounds
else
    %50 or more moves were played
    if M==51
        move=Defect; %defect on 51st move
    else %51 or more moves were played
        if (51 < M) && (M < 57) %for moves 52−56 plays TFT
            move=TFT(My_hist, Opp_hist); %play TFT for
                                         %the next 5 moves
        else %56 or more were played
            if ISRANDOM(Opp_hist) %if opponent plays
                                  %random strategy
                move = Defect;
            else %if opponent does not play a random strategy
                if DO_I_KNOW_THIS_STRATEGY(My_hist, Opp_hist)
                    %if opponent's strategy is recognized
                    move = TFT(My_hist, Opp_hist);
                else
                    %if opponent's stategy is not recognized
                    %we code it as defect randomly with
                    %probability 10%
                    if rand()≤0.1  %defect 10% of the time
                        move = Defect;
                    else %cooperate 90% of the time
                        move = Cooperate;
                    end;
                end;
```

```
                    end;
                end;
            end;
        end;
    end

    function move=Joss(My_hist, Opp_hist)
        %Joss Strategy
        %Plays a variation of TFT; it always defects when the
        %opponent defects, but it cooperates when the opponent
        %cooperates with a probability of .9

        if isempty(My_hist) %if first move
            move=Cooperate;
        else %if not first move
            if Opp_hist(end)==Cooperate %if the opponent
                                        %cooperated on last move
                if 0.9 ≤ rand()
                    move=Cooperate; %cooperate with a probability
                                    %of 0.9
                else
                    move=Defect; %defect 10% of the time when the
                                 %opponent cooperates
                end;
            else %if the opponent defected on the last move
                move=Defect;
            end;
        end;
    end
```

```
function output = A_SCORE(My_hist, Opp_hist)
    %Implements the function: A = 16a1 + 4a2 + a3
    %a1 is the score from the previous round
    %a2 is the score from 2 moves before
    %a3 is the score from 3 moves before
    %ai = 3 if both defect
    %ai = 2 if only the opponent defects
    %ai = 1 if only it defects
    %ai = 0 if both cooperate

    %%This function will be used within the Nydegger strategy
    %below.

    score_map =[3,1;  %score map matrix
        2,0];
    %if both players defect, score 3 points
    %if I defect and opponent cooperates, score 1 point
    %if I cooperate and opponent defects, score 2 points
    %if both players cooperate score 0 point

    A_SCORE = 16*score_map(My_hist(end), Opp_hist(end)) + ...
        4*score_map(My_hist(end - 1), Opp_hist(end-1)) + ...
        score_map(My_hist(end - 2), Opp_hist(end-2));
    %implement the score function
    output = A_SCORE;
end

function move=Nydegger(My_hist, Opp_hist)
```

```
%Nydegger Strategy
%Plays a variation of TFT for 3 rounds: if it is the only
%one to cooperate on first round, and only one to defect on
%second round, then then it defects on round 3. After first
%3 moves, the following moves are based on the previous 3
%rounds based on a score given by making a calculation:
%A = 16a1 + 4a2 + a3, where ai is the score for the
%previous ith round, ai = 3 if both strategies defect,
%ai=2 if only the opponent defects, and ai = 1 if only it
%defects. The strategy defects if and only if
%A = {1, 6, 7, 17, 22, 23, 26, 29, 30, 31, 33,38, 39, 45,
%49, 54, 55, 58, 61}

M = length(My_hist) + 1; %denotes the current round
if M ≤ 2 %for the first 2 moves
    move = TFT(My_hist, Opp_hist);
else %if more than 2 moves have been played
    if M==3 %on the third move
        %if it is the only one to
        %cooperate on first round, and only one to defect
        %on second round, then it defects
        if (My_hist(end−1) == Cooperate) && ...
            (Opp_hist(end−1) == Defect) ...
                && (My_hist(end) == Defect) && ...
                    (Opp_hist(end) == Cooperate)
            move = Defect;
            %I am the only one to cooperate in the first
            %round and I am the only one to defect in
            %the second round
```

```
                else
                    move = Cooperate;
                end;

            else %if more than 3 moves are played
                a = [ 1 6 7 17 22 23 26 29 30 31 33 38 39 45 49 ...
                    54 55 58 61];
                %possible output values from the A Score function
                if ismember(A_SCORE(My_hist, Opp_hist), a) == 1
                %defect if the A Score is one of the scores in "a"
                    move = Defect;
                else %if A_Score is not one of those values in "a"
                    move = Cooperate;
                end
            end
        end
    end

    function move=Shubik(My_hist, Opp_hist)
        %Shubik Strategy
        %Plays a variation of TFT.  It cooperates when the
        %opponent cooperates, and it begins with a single
        %defection if the opponent defects.  But,the retaliation
        %increases by 1 each time the opponent defects when it had
        %cooperated on the previous round.

        if isempty(My_hist) %if first move
            move=Cooperate; %cooperate on first move
            Retaliation_Counter = 0; % init the count of retaliation
```

```
            Moves_to_retaliate = 0; % not retaliating anymore

        else %if not first move
            if Moves_to_retaliate > 0
                %if I am retaliating
                %ignore opponent completely and defect
                %this has to go for a total of
                %Retaliation_Counter moves. It is done by
                %an auxiliary counter Moves_to_retaliate that
                %decreases by 1 every time we defect
                move = Defect;
                Moves_to_retaliate = Moves_to_retaliate - 1;
                %decrease the number of moves I still have to
                %retaliate
            else %I am not retaliating
                if (Opp_hist(end) == ...
                    Defect)&&(My_hist(end)==Cooperate)
                    %this means unprovoked defection
                    %I have to start retaliating
                    move = Defect;
                    Moves_to_retaliate = Retaliation_Counter;
                    %how many more moves I have to retaliate
                    Retaliation_Counter = Retaliation_Counter + 1;
                    %next time I will retaliate one move longer
                else
                    move = Cooperate;
                end;
            end;
        end;
```

```
    end

    function move=Stein(My_hist, Opp_hist)
        %Stein and Rapoport Strategy
        %This strategy plays a modification of Tit For Tat
        %Cooperates for first 4 moves, then plays TFT, checking
        %every 15 moves to see if the opponent is playing randomly.
        %If the opponent is playing randomly, it defects.  Otherwise,
        %it cooperates.
        %It defects on last 2 moves.


        M = length(My_hist) + 1; %denotes the current round
        if M ≤ 4 %for the first 4 moves
            move=Cooperate; %Cooperate for the first 4 moves
            Stein_move_counter = 0; %initialize the move counter
            Opponent_is_random = 0; %assume opponent is not random
        else %if more than 4 moves have been played
            if (4 < M) && (M < 199)
                Stein_move_counter = Stein_move_counter +1;
                %increase the move counter
                if (Stein_move_counter ==15)
                    %every 15 moves, check if opponent
                    %is random
                    Opponent_is_random = ISRANDOM(Opp_hist);
                    Stein_move_counter = 0; %reset the counter
                end
                if Opponent_is_random == 1 %if my opponent is random
                    move = Defect;
```

```
            else %if my opponent is not random
                move = TFT(My_hist, Opp_hist);
                %play TFT for all rounds up
                %until last 2 moves
            end
        end
    end
    if M ≥ 199 %for the last 2 moves
        move = Defect;
    end
end

function move=Downing(My_hist, Opp_hist)
    %Revised Downing Strategy
    %In the original tournament, Downing defected on the
    %first two moves.  This is corrected and we implement
    %the Revised Downing strategy.

    %It calculates conditional probability that the opponent
    %will cooperate given that it defected and the conditional
    %probability that the opponent will cooperate given that it
    %cooperated. If the opponent seems unresponsive to what
    %it is doing, it will defect as much as possible. If the
    %opponent seems responsive, it cooperates. It uses these
    %probabilities to estimate the opponent's next move. These
    %probabilities are continuously updated and the strategy
    %attempts to make moves that will maximize the score on
    %the long term.

```

```
        if isempty(My_hist) %if first move
            move = Cooperate; %cooperate on first move
            %initialize counters
            good = 1;
            bad = 0;
            C_count = 0;   %count of my own cooperations
            D_count = 0;   %count of my own defections
            DC_count = 0;  %count of opponent cooperations after
                           %my defection
            CC_count = 0;  %count of opponent cooperations after
                           %my cooperation
        else
            if length(My_hist)<2 %if 2nd move
                move = Cooperate; %cooperate on the 2nd move too
            else %third move or more
                if My_hist(end) == Defect %if I defected on the
                                           %last move
                    D_count = D_count + 1; %increase the count
                                           %of my defections
                    if Opp_hist(end) == Cooperate %if opponent
                                                  %cooperated
                                                  %despite my
                                                  %defection
                        DC_count = DC_count + 1;
                    end
                    bad = DC_count/D_count; %update the
                                            %probability
                                            %of the opponent
                                            %cooperating despite
```

```
                                    %a defection
                %This is the probability that the
                %opponent is unresponsive.
            else %if I cooperated
                C_count = C_count + 1; %increase the count of
                                        %my cooperations
                if Opp_hist(end) == Cooperate %if the
                                              %opponent
                                              %cooperated
                                              %following
                                              %my cooperation
                    CC_count = CC_count + 1;
                end
                good = CC_count/C_count; %update the
                                         %probability
                                         %of the opponent
                                         %cooperating following
                                         %a cooperation
                %This is the probability that the
                %opponent is responsive.
            end
            %Next, make a decision based on the updated
            %probabilities.
            c = 6.0*good - 8.0*bad - 2;
            alt = 4.0*good -5.0*bad - 1;

            if c ≥ 0 && c ≥ alt %if opponent seems responsive
                move= Cooperate;
            else
```

```
                if (c ≥ 0 && c < alt) || (alt ≥ 0)
                    move=3−My_hist(end); %do the opposite of
                                         %my last move
                else %if the opponent doesn't seem responsive
                    move = Defect;
                end
            end
        end
    end
end

function move=Feld(My_hist, Opp_hist)
    %Feld Strategy
    %Plays TFT in that it begins with a cooperation and
    %defects every time the opponent defects, but it
    %cooperates with a decreasing probability until it
    %reaches 0.5. We decrease the probability each time by
    %0.05

    if isempty(My_hist) %if first move
        move=Cooperate;
        probability_to_cooperate = 1;
    else %if not first move
        if Opp_hist(end)==Defect %and it defected on the last
                                 %move
            move=Defect;
        else %opponent cooperates
            if probability_to_cooperate ≥ rand() %cooperate
                                                  %with a given
```

```
                                                %probability
                move=Cooperate;
            else %defect the other 0.5 of the time
                move=Defect;
            end;
            %decrease the probability to cooperate by 0.05
            %but always keep it at least 0.5
            probability_to_cooperate = max(0.5, ...
                probability_to_cooperate−0.05);
        end;
    end;
end

function move=Tullock(My_hist, Opp_hist)
    %Tullock Strategy
    %Cooperates the first 11 rounds, and then randomly
    %cooperates 10% less than the opponent cooperated
    %in the previous 10 rounds

    if length(My_hist) < 11 %if less than 11 rounds have been
                            %played
        move=Cooperate;
    else %if more than 11 rounds have been played
        Opp_last_10_moves = Opp_hist(end−10+1:end); %get the
                                                    %last 10
                                                    %moves
        prob_to_coop = ...
            max(0,sum(Opp_last_10_moves==Cooperate)/10 − 0.1);
        if rand()≤prob_to_coop
```

```
                move=Cooperate;
            else
                move = Defect;
            end;
        end;
    end

    function move=Unnamed(My_hist, Opp_hist)
        %Unnamed Strategy
        %It cooperates with a given probability P.  This
        %probability is initially 0.3.  Then P is updated
        %every 10 rounds based on whether the opponent
        %seems very random, very cooperative, or very
        %uncooperative. Also, after 130 rounds, P is adjusted
        %if it is losing to the opponent.

        %The original code is not available, and has been deemed
        %"complicated" but based on public descriptions, it can
        %be determined that this strategy cooperates with a random
        %probability between 0.3 and 0.7

        random_number=rand(); %generate random number
        if 0.3<random_number && 0.7>random_number  %for a
                                                   %probability
                                                   %between 0.3
                                                   %and 0.7
            move=Cooperate;
        else %in the other 0.3 to 0.7 of the time
            move=Defect;
```

```
        end;
    end

    function score = get_score(My_hist, Opp_hist)
        %returns my current total score in the IPD Game
        %checks every round of the game, collects scores in that
        %round and adds them up

        %This function is used in the Tideman strategy below.

        score = 0; %initialize the counter of the score
        if isempty(My_hist)
            score = 0;
        else
            for i=1:length(My_hist)
                score = score + PD_payoff(My_hist(i), Opp_hist(i));
            end
        end
    end

    function move=Tideman(My_hist, Opp_hist)
        %Tideman and Chieruzzi Strategy
        %It plays the Shubik Strategy with a slight variation.
        %The opponent is given a "fresh start" if certain criteria
        %are met:
        %   1. The opponent is 10 points behind this strategy
        %   2. AND if the opponent has not just begun a run of
        %   defections
        %   3. AND if it has been at least 20 rounds since the
```

```
        %   last "fresh start"
        %   4. AND there are 10 or more rounds left in the tournament
        %   5. AND the  total number of defections differs from a
        %   50-50 random sample by at least 3.0 standard deviations.
        %A "fresh start" is a sequence of 2 cooperations and an
        %assumption that the game has just started (so all is
        %forgotten)

        last_refresh_round = -20; % init of the counter keeping
                                   %track of last refreshing, has
                                   %to be -20 to make sure we can
                                   %refresh soon if needed

        if isempty(My_hist) %if first move
            move=Cooperate; %cooperate on first move
            Retaliation_Counter = 0; % init the count of retaliation
            Moves_to_retaliate = 0; % not retaliating anymore

        else %if not first move
            %check if I should restart the counter
            %   1. The opponent is 10 points behind this strategy
            %   2. AND if the opponent has not just begun a run
            %   of defections
            %   3. AND if it has been at least 20 rounds since
            %   the last "fresh start"
            %   4. AND there are 10 or more rounds left in the
            %   tournament
            %   5. AND the  total number of defections differs
            %   from a 50-50 random sample by at least 3.0
```

```
        %   standard deviations.
        %get the scores
        My_score = get_score(My_hist, Opp_hist); % get the
                                                  %score for
                                                  %me based on
                                                  %the history
        Opp_score = get_score(Opp_hist, My_hist); %get the
                                                   %score for
                                                   %me based on
                                                   %the history
        current_round = length(My_hist)+1; %round number to
                                           %be played
        n = length(My_hist); %number of rounds already played
        opp_D_counter = sum(Opp_hist==Defect); %count of
                                               %opponent's
                                               %defection so far
        if (My_score - Opp_score ≥ 10) ...
                && (Opp_hist(end)==Cooperate) ...
                && (current_round-last_refresh_round≥ 20) ...
                && (current_round ≤ 190) ...
                && ((opp_D_counter < (n/2 - ...
                   3*sqrt(n/4)))||(opp_D_counter > (n/2 + ...
                   3*sqrt(n/4))))
            %now I can give a fresh start
            Retaliation_Counter = 0; % init the count of
                                     %retaliation
            Moves_to_retaliate = 0; % not retaliating anymore
            last_refresh_round = current_round; % I just
                                                %refreshed,
```

```
                                                        %so I need to
                                                        %store the info
            move = Cooperate;
        else
            %I did not give a fresh start, so I am doing Shubik
            %I can't call Shubik because of internal counting
            %in this procedure
            if Moves_to_retaliate > 0
                %if I am retaliating
                %ignore opponent completely and defect
                %this has to go for a total of
                %Retaliation_Counter moves. It is done by
                %an auxiliary counter Moves_to_retaliate that
                %decreases by 1 every time we defect
                move = Defect;
                Moves_to_retaliate = Moves_to_retaliate - 1;
                %decrease the number of moves I still have
                %to retaliate
            else %I am not retaliating
                if (Opp_hist(end) == ...
                   Defect)&&(My_hist(end)==Cooperate)
                    %this means unprovoked defection
                    %I have to start retaliating
                    move = Defect;
                    Moves_to_retaliate = Retaliation_Counter;
                    %how many more moves I have to retaliate
                    Retaliation_Counter = Retaliation_Counter ...
                        + 1;
                    %next time I will retaliate one
```

```
                        %move longer
                    else
                        move = Cooperate;
                    end;
                end;
            end;
        end;
    end
%

%%Sample Play

    function [hist1, hist2]=SamplePlay(Strat1,Strat2, n_of_moves)
        % produces two histories for a game of n_of_moves rounds
        %of strategy Strat1 playing against strategy Strat2

        aux_hist1=[]; %initialize auxiliary histories
        aux_hist2=[]; %initialize auxiliary histories
        for round=1:n_of_moves
            move1 = strategy{Strat1}(aux_hist1, aux_hist2);
            %move for player 1
            move2 = strategy{Strat2}(aux_hist2, aux_hist1);
            %move for player 2
            aux_hist1 = [aux_hist1, move1];
            %update history of player 1
            aux_hist2 = [aux_hist2, move2];
            %update history of player 2
        end;
        hist1 = aux_hist1;
```

```
        hist2 = aux_hist2;
    end


%% Actual tournament (round robin)

    function [score1, score2]=Axelrod(Strat1,Strat2, n_of_moves)
        % produces two histories for a game of n_of_moves rounds
        % of strategy Strat1 playing against strategy Strat2
        % also produces two scores from a game of n_of_moves rounds

        aux_hist1=[]; %initialize auxiliary histories
        aux_hist2=[]; %initialize auxiliary histories
        P1score = 0; %initialize player 1 score
        P2score = 0; %initialize player 2 score
        for round=1:n_of_moves
            move1 = strategy{Strat1}(aux_hist1, aux_hist2);
            %move for player 1
            move2 = strategy{Strat2}(aux_hist2, aux_hist1);
            %move for player 2
            aux_hist1 = [aux_hist1, move1];
            %update history of player 1
            aux_hist2 = [aux_hist2, move2];
            %update history of player 2
            P1score = P1score + PD_payoff(move1, move2);
            %update score of player 1
            P2score = P2score + PD_payoff(move2, move1);
            %update score of player 2
        end;
```

```
        score1 = P1score;
        score2 = P2score;
    end

%%Display Outcomes from Actual Axelrod Tournament

for k = 1:1000 %play the tournament 100 times
    display(['playing round ' num2str(k)])
    for Strat1 = 1:15       % all players will play
        for Strat2 = Strat1:15  % with every other player
            [score1, score2]=Axelrod(Strat1, Strat2, 200);
            SCORES_OUTPUT(Strat1, Strat2) = score1;
            SCORES_OUTPUT(Strat2, Strat1) = score2;
        end
        SCORES_OUTPUT(Strat1, 16) = mean(SCORES_OUTPUT(Strat1, ...
           1:15));
    end


    total_scores = SCORES_OUTPUT(:, 16);

    [¬, indices]  = sort(total_scores, 'descend');



    %AUTOMATICALLY INCLUDE THE ORDER
    for ii=1:15
        SCORES_OUTPUT(indices(ii),17) = ii;
    end

```

```
    %this is for number outputs only
    xlswrite('number_outputfile', SCORES_OUTPUT, k);

    %uncomment the things below for getting nice tables
    %             SCORES_TO_WRITE(2:16, 2:18) = SCORES_OUTPUT;
    %             xlswrite('outputfile', SCORES_TO_WRITE, k);
    %             TABLE(1, 2:16) = strategy_names;
    %             TABLE(1, 17:18) = {'Average', 'Order'};
    %             xlswrite('outputfile', TABLE, k);
    %             TABLE2(2:16,1) = strategy_names;
    %             xlswrite('outputfile', TABLE2, k);

end
end
```

A.2 Matlab Code for Data Analysis for Tournament

```
function data_analysis
%reads outputs generated by IPD Axelrod Tournament and
%analyzes it.
%It collects the best, worst, and average scores from each
%of the sheets in the outputfile from the IPD_Tournament.

strategy_names = {'TFT', 'Tideman', 'Nydegger', 'Grofman', ...
    'Shubik', 'Stein', 'Grim', 'Davis', 'Graaskamp', 'Downing', ...
    'Feld', 'Joss', 'Tullock', 'Unnamed' 'Random'};


```

```
number_of_sheets = 1000;
%read the output file into one single variable
for sheet=1:number_of_sheets
    display(['now reading sheet ' num2str(sheet)])
    output(:,:,sheet) = xlsread('outputfile.xls',sheet);
end


for row = 1:15
    for column = 1:17
        aux = output(row,column,:);
        if column <17
            best_score(row,column) = max(aux);
            %looking for the maximum score
            worst_score(row,column) = min(aux);
            %looking for the minimum score
            average_score(row,column) = mean(aux);
            %looking for the average score
        else
            best_score(row,column) = min(aux);
            %looking for the highest place
            worst_score(row,column) = max(aux);
            %looking for the lowest place
            average_score(row,column) = mean(aux);
            %looking for the minimum score
        end
    end
end

```

```
    function write_it_nicely(input, filename)
        %writes input matrix into a nice table with the headings
        %into the specified file
        TO_WRITE(2:16, 2:18) = input;
        xlswrite(filename, TO_WRITE);
        TABLE(1, 2:16) = strategy_names;
        TABLE(1, 17:18) = {'Average', 'Order'};
        xlswrite(filename, TABLE);
        TABLE2(2:16,1) = strategy_names;
        xlswrite(filename, TABLE2);
    end

write_it_nicely(best_score,'best.xls')
write_it_nicely(worst_score,'worst.xls')
write_it_nicely(average_score,'average.xls')
end
```

APPENDIX B

DESCRIPTIONS OF STRATEGIES IN AXELROD'S TOURNAMENT

Here we include a description of each of the strategies that competed in Axelrod's original computer tournament. Any variations that were implemented in our computer tournament are also indicated in the description. The descriptions listed here were compiled using information in the Axelrod Library [KCH+17].

(1) **TIT FOR TAT.** Always cooperates on the first move. After the first move, it reciprocates the opponent's last move.

(2) **TIDEMAN.** It plays the Shubik Strategy with a slight variation. The opponent is given a "fresh start" if certain criteria are met:

   (a) The opponent is 10 points behind this strategy

   (b) AND if the opponent has not just begun a run of defections

   (c) AND if it has been at least 20 rounds since the last "fresh start"

   (d) AND there are 10 or more rounds left in the tournament

   (e) AND the total number of defections differs from a 50-50 random sample by at least 3.0 standard deviations.

   A "fresh start" is a sequence of 2 cooperations and an assumption that the game has just started (so all is forgotten).

(3) **NYDEGGER.** Plays a variation of TFT for 3 rounds: if it is the only one to cooperate on first round, and only one to defect on second round, then then it defects on round 3. After first 3 moves, the following moves are based on the

previous 3 rounds based on a score given by making a calculation:

$A = 16a_1 + 4a_2 + a_3$, where a_i is the score for the previous ith round:

(a) $a_i = 3$ if both strategies defect.

(b) $a_i = 2$ if only the opponent defects.

(c) $a_i = 1$ if only it defects.

The strategy defects if and only if A = {1, 6, 7, 17, 22, 23, 26, 29, 30, 31, 33,38, 39, 45, 49, 54, 55, 58, 61}.

(4) **GROFMAN.** It cooperates on the first two moves, and then returns the opponent's moves for the next five moves (i.e. It cooperates on the first move and then plays TFT for moves 2-6). For the remaining moves of the game, it cooperates if both it and the opponent made the same move in the previous round. Otherwise, it cooperates randomly with a probability of 2/7.

(5) **SHUBIK.** Plays a variation of TFT. It cooperates when the opponent cooperates, and it begins with a single defection if the opponent defects. But,the retaliation increases by 1 each time the opponent defects when it had cooperated on the previous round.

(6) **STEIN & RAPOPORT.** This strategy plays a modification of TIT FOR TAT. It cooperates for first 4 moves, then plays TFT, checking every 15 moves to see if the opponent is playing randomly. If the opponent is playing randomly, it defects. Otherwise, it cooperates. Finally, it defects on last 2 moves.

(7) **FRIEDMAN.** This strategy will cooperate until the opponent defects. Then, it will always defect for all of the remaining moves.

(8) **DAVIS.** This strategy cooperates on the first 10 moves, then it plays FRIEDMAN for the remaining moves of the game.

(9) **GRAASKAMP.** Plays TFT for 50 rounds, defects on round 51, plays TFT for rounds 52-56, a check is then made to see if the opponent is playing randomly, if so it defects for the rest of the rounds. The strategy also checks to see if the opponent is playing some other strategy that it recognizes. If so, it plays TFT for the remaining moves of the game.Otherwise, if the opponent is not playing a recognizable strategy, it cooperates and randomly defects every 5 to 15 moves. The last bit is coded by randomly defecting with probability 0.1.

(10) **DOWNING.** In the original tournament, DOWNING defected on the first two moves. This is corrected and we implement the REVISED DOWNING strategy. It calculates the conditional probability that the opponent will cooperate given that it defected and the conditional probability that the opponent will cooperate given that it cooperated. If the opponent seems unresponsive to what it is doing, it will defect as much as possible. If the opponent seems responsive, it cooperates. It uses these probabilities to estimate the opponent's next move. These probabilities are continuously updated and the strategy attempts to make moves that will maximize the score on the long term.

(11) **FELD.** This strategy plays TFT in that it begins with a cooperation and defects every time the opponent defects, but it cooperates with a decreasing probability until it reaches 0.5. We decrease the probability each time by 0.05.

(12) **JOSS.** It plays a variation of TFT. It always defects when the opponent defects, but it cooperates when the opponent cooperates with a probability of .9.

(13) **TULLOCK.** Cooperates the first 11 rounds, and then randomly cooperates 10% less than the opponent cooperated in the previous 10 rounds.

(14) **UNNAMED.** It cooperates with a given probability P. This probability is initially 0.3. Then P is updated every 10 rounds based on whether the opponent seems very random, very cooperative, or very uncooperative. Also, after 130 rounds, P is adjusted if it is losing to the opponent. The original code is not available, and has been deemed "complicated," but based on public descriptions, it can be determined that this strategy cooperates with a random probability between 0.3 and 0.7.

(15) **RANDOM.** Cooperates and defects on a completely random basis–not dependent on the opponent's moves.